LEADER AND FOLLOWERS

Leader and Followers

- Intento: Avere un singolo server che coordina la replicazione per tutto un insieme di server
- Problema
  - Per ottenere la fault tolerance in sistemi che gestiscono dati, i dati devono essere replicati su più server
  - Bisogna anche fornire garanzie di consistenza ai client. Quando i dati vengono aggiornati su vari server, occorre decidere quando rendere visibili gli aggiornamenti ai client
  - Un ***Majority Quorum*** (quorum di maggioranza) non è sufficiente, poiché in alcuni scenari di fallimento dei server i client vedrebbero dati inconsistenti. Solo quando i dati sono letti da più server le inconsistenze possono essere risolte, ma a volte questo non basta

Quorum di Maggioranza

- Si potrebbe pensare che raggiungere un quorum di lettura o scrittura sia sufficiente a garantire consistenza forte in caso di fallimento del server, ma non è così
- Supponiamo di avere tre server (e un fattore di replica pari a tre). Inizialmente la variabile x vale 1
  - Lo scrittore1 scrive x = 2 e la richiesta di scrittura viene mandata ai tre server. La scrittura ha successo sul server1, ma fallisce sui server2 e server3
  - Il client c1 legge il valore di x da server1 e server2 e prende il valore di x = 2 poiché il server1 ha l'ultimo valore
  - Il client c2 avvia la lettura di x, ma il server1 va giù. Quindi c2 legge da server2 e server3 che hanno x pari a 1. Quindi c2 legge il vecchio valore di x sebbene la lettura avviene dopo che c1 ha letto il valore recente di x

Leader and Followers

- Soluzione
  - Selezionare un server nel cluster come Leader. Il Leader è responsabile prende decisioni per l'intero cluster e propaga le decisioni a tutti i server del cluster
  - Ogni server all'avvio cerca un leader, se non lo trova allora avvia una elezione del leader. I server accettano le richieste solo dopo che un leader è stato eletto
  - Solo il leader gestisce le richieste dei client. Se una richiesta è mandata direttamente a un server che è un follower, questo la propaga al leader

Leader and Followers

- Elezione del Leader: premesse
  - Ogni server, all'avvio fa partire una elezione del leader
  - Finché il leader non viene eletto non si accettano richieste
  - A ogni elezione del leader si incrementa il numero della **generazione** (***Generation Clock***)
  - Un server può essere solo in uno degli stati: Leader, Follower, Cerca Leader (detto pure Candidato)
  - Si usa il meccanismo di ***Heart Beat*** per determinare se un precedente leader non funziona più, e quindi per far partire una nuova elezione del leader

HEART BEAT

Leader and Followers

- Elezione del Leader: tecniche
  - Il server più aggiornato può essere il leader. Il server più aggiornato è quello che ha il Generation Clock più recente, e il più recente indice di log in *Write-Ahead Log*
  - Se tutti i server sono equamente aggiornati allora si usa uno dei seguenti criteri per la scelta del leader
    - Ogni server ha un identificativo (ID), si vota il server con ID più alto
    - Ogni server chiede il voto in tempi diversi (random), vince il server che avvia l'elezione per primo (algoritmo RAFT, Reliable Replicated Redundant and Fault-Tolerant)
  - Lo stesso voto che è stato dato per un Generation Clock viene restituito in caso si ha una nuova richiesta di elezione con lo stesso Generation Clock
  - Il leader è eletto se ottiene la maggioranza dei voti

Heart Beat

- Intento: mostrare che un server è disponibile mandando periodicamente un messaggio a tutti gli altri server
- Problema
  - Quando più server formano un cluster, ogni server è responsabile per immagazzinare una porzione di dati, in base agli schemi di partizionamento e replica usati
  - Accorgersi prontamente di un fallimento di un server è importante per avviare azioni correttive rendendo un server responsabile per la gestione delle richieste dei dati che erano sul server non disponibile

Heart Beat

- Soluzione
  - Mandare un *richiesta periodica* agli altri server che indica la presenza del server che invia la richiesta (**liveness**)
  - Selezionare l'*intervallo di richiesta* in modo che sia maggiore del *tempo di round-trip* (tempo di andata e ritorno) fra i server
  - I server aspettano un *intervallo di timeout*, che è un multiplo dell'intervallo di richiesta
  - Intervallo di timeout > intervallo di richiesta > intervallo round-trip
  - Es.: round-trip = 20 ms, heartbeat = 100ms, timeout = 1s
  - Un intervallo di heartbeat piccolo permette di determinare velocemente un fallimento, ma vi è una probabilità più alta di falsi rilevamenti di fallimento

MAJORITY QUORUM

Majority Quorum

- Intento: Evitare di avere due gruppi di server che prendono decisioni indipendenti tramite la richiesta di una maggioranza per ciascuna decisione presa
- Problema
  - Safety e Liveness. **Liveness** è la proprietà di un sistema che dice che il sistema progredisce sempre. **Safety** è la proprietà che dice che il sistema è sempre nello stato corretto
  - Quando un server aggiorna un dato, la replica del dato su altri server assicura che il dato sarà disponibile pur se il server fallisce. Quante conferme di copie del dato devono essere ricevute? Con *tante copie* ci vorrà più tempo per le conferme e si riduce la *liveness*, *troppe poche copie* e il dato potrebbe essere perso e non si ha *safety*

Majority Quorum

- Soluzione
  - Un cluster concorda di aver ricevuto un aggiornamento quando la *maggioranza dei nodi* del cluster ha confermato l'aggiornamento. Tale numero è chiamato quorum. Se un cluster ha n nodi, il quorum è n/2 + 1
    - Un cluster con cinque nodi avrà un quorum pari a 3
  - Il quorum indica quanti fallimenti possono essere tollerati. Il numero di fallimenti tollerati è la *dimensione del cluster meno il quorum*
    - Se si vogliono tollerare f fallimenti si necessita di un cluster di dimensioni 2f + 1

Majority Quorum

- In un cluster che replica i dati si dovrà considerare
  - *Il throughput delle operazioni di scrittura*. Ogni scrittura deve essere replicata su più server, ogni server aggiunge un rallentamento. Complessivamente il rallentamento è proporzionale al numero di server che formano il quorum. Quindi raddoppiare il numero di server significa ridurre il throughput di metà
  - *Il numero di fallimenti che devono essere tollerati*. Il numero di fallimenti di server tollerati dipende dalla dimensione del cluster. L'aggiunta di un server non sempre migliora la tolleranza ai fallimenti
    - Con 3 server, il quorum è 2, il numero di fallimenti tollerati è 1
    - Con 4 server, il quorum è 3, il numero di fallimenti tollerati è 1
    - Con 5 server, il quorum è 3, il numero di fallimenti tollerati è 2

GENERATION CLOCK

Majority Quorum

- Quorum flessibili
  - Due operazioni diverse potrebbero avere quorum di dimensioni diverse a condizione che vi è una intersezione
  - Un'operazione frequente potrebbe avere un quorum di dimensione più piccolo per renderla più veloce. Tipicamente le operazioni di lettura sono più frequenti
  - In un cluster di 5 nodi, il quorum di lettura potrebbe essere 2 mentre il quorum di scrittura 4. L'intersezione è quindi non vuota

Generation Clock (Epoca, Generazione)

- Intento: un numero che cresce monotonicamente e che indica la generazione di un server
- Problema
  - In un contesto di Leader e Follower, c'è la possibilità che un leader sia temporaneamente disconnesso dai follower per mancanza di rete. Il processo del leader esegue ancora, e quando la rete ritorna disponibile il server manderà richieste di repliche ai follower
  - Tuttavia nel frattempo il cluster potrebbe aver selezionato un nuovo leader e aver servito le richieste da client
  - Il cluster deve poter rilevare se le richieste provengono da un vecchio leader. Il vecchio leader dovrebbe poter rilevare se è stato momentaneamente disconnesso dal cluster

Generation Clock

- Soluzione
  - Tenere un numero che si incrementa e che indica la generazione del server
  - Ogni volta che si ha un'elezione di un leader, questa è marcata con un incremento della generazione
  - La generazione deve essere disponibile a un riavvio del server. All'avvio il server legge l'ultima generazione conosciuta dal log
  - I server di ***Leader e Follower*** incrementano la generazione ogni volta che vi è una nuova elezione del leader
  - I server mandano la generazione agli altri server insieme alla richiesta di voto. Quando si completa l'elezione del leader, tutti i follower sono avvisati della nuova generazione
  - Il leader include la generazione in ogni richiesta ai follower, e nel messaggio ***Heart Beat***

Generation Clock

- Esempio
  - In un cluster con tre server, Leader1 è il leader esistente e la generazione è 1. Il Leader1 manda i messaggi di heart beat con generazione 1
  - Il Leader1 si ferma per 5 secondi, nel frattempo i server eleggono il nuovo leader e la generazione è 2
  - Il Leader1 dopo la pausa manda richieste ai server, ma i follower e il nuovo leader respingono le richieste e mandano una risposta di fallimento, poiché la generazione è 2
  - Il Leader1 riceve le risposte di fallimento e diventa un follower con generazione aggiornata a 2